Universidad Nacional de Rosario

Facultad de Cs. Exactas, Ingeniería y Agrimensura

Procesamiento de Imágenes Digitales

Trabajo Práctico N°3

López Ceratto, Julieta : L-3311/1,
Slepoy, David : S-5782/7

Diciembre, 2024

Figure 1: Logo Facultad Cs. Exactas, Ingeniería y Agrimensura

Introducción

En este informe se explicará de manera detallada los procedimientos realizados y resultados obtenidos para el ejercicio planteado por la cátedra. Se abordarán tanto el planteo de la solución como el código realizado y la lógica detrás del mismo.

Contents

Chapter 1

Generala

1.1 Breve explicación

La Generala es un juego de dados que se juega con cinco dados y generalmente entre dos o más jugadores. El objetivo es lograr la mayor cantidad de puntos posibles combinando tiradas de dados en categorías específicas, como escalera, full, póker o generala (cinco dados iguales). En si, las jugadas son:

- Escalera: secuencia consecutiva de números.
- Full: combinación de tres dados iguales y dos dados iguales (un trío y un par)
- Póker: tener cuatro dados iguales y un dado distinto.
- Generala: combinación máxima, donde los cinco dados tienen el mismo valor.

Cada jugador tiene tres oportunidades por turno para lanzar los dados y elegir qué combinación intentar. Se van anotando los resultados en una hoja de puntuación, y el juego termina cuando todas las categorías están llenas. Gana quien acumule más puntos.

1.2 Planteo del problema

Se plantean 4 videos, cada uno de ellos representa una tirada de dicho juego. El problema a resolver es decir qué jugada obtuvo el jugador y, finalmente, los puntos obtenidos. Para cada video, debe enmarcase los dados en un rectángulo, cuando los mismos se quedan quietos, contar los puntos y mostrarlos. Se debe devolver un video de cada tirada; el mismo debe tener los dados enmarcados, los puntos de cada uno, los puntos totales de la tirada. Además se debe imprimir por pantalla los dados y sus puntos, junto con los puntos totales.

1.3 Abordaje

Para este problema se abordaron los siguientes pasos:

1. Detectar área de interés.
2. Detectar frame donde los dados se dejan de mover.
3. Detectar los dados de todos los frames.
4. Detectar los puntos de los dados en el frame donde se detienen los dados.

5. Detectar jugada en el frame donde se detienen los dados.

6. Dibujar cada frame según corresponda y mostrar o guardar el video.

1.3.1 Punto 1: Detectar área de interés.

El área de interés, en el contexto de este problema, se define como aquella área que abarca el tablero verde donde se produce la jugada. Es de suma importancia ya que todo el análisis a posteriori debe realizarse sobre esta área: detección de movimiento, dados, puntaje, entre otros. Como primer paso, se obtiene un cuadrado de muestra recortado de forma manual para el primer frame del video; de esta manera se obtiene una muestra del color verde a segmentar:
Se aplica el delta [10.,90.,90.] a la muestra para obtener los límites inferior y superior:

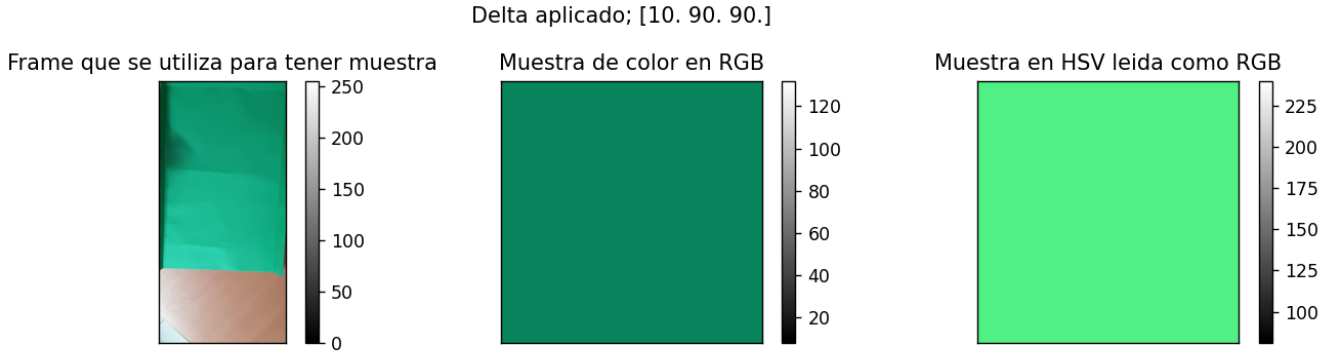

Figure 1.1: Muestra de color obtenida

- Límite inferior: [[[71. 150. 42.]]]
- Límite superior: [[[91. 255. 222.]]]

Todo esto, lo hacemos con la siguiente función:

```
def green_sample(frame)-> tuple[np.array, np.array,np.array]:
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    bg_crop_RGB = frame_rgb[100:200, 100:200, :]
    #mshow(bg_crop_RGB)                    # Cortar una
porcion correspondiente al fondo.
    #imshow(bg_crop_RGB, title="Frame 1 - Crop fondo verde")
    img_pixels = bg_crop_RGB.reshape(-1,3)
    colours, counts = np.unique(img_pixels, axis = 0,
return_counts= True)
    n_colours = colours.shape[0]
    n_colours
    colours[0]

    bg_color_rgb = colours[4]
    bg_color_rgb = np.reshape(bg_color_rgb, (1,1,-1))
    bg_color_hsv = cv2.cvtColor(bg_color_rgb,
cv2.COLOR_RGB2HSV)
    DELTA_BG = np.array([10.0, 90.0, 90.0])
    #Define limite inferior y superior en rango de verde.
    lower_limmit = np.clip(bg_color_hsv - DELTA_BG, 0, 255)
    upper_limmit = np.clip(bg_color_hsv + DELTA_BG, 0, 255)
    return lower_limmit, upper_limmit
```

Figure 1.2: Función green sample

Con dicho rango de color verde se obtiene el área de interés. Esto se hace aplicando al frame de test (primer frame del video del cual se obtiene la muestra) una máscara que contiene True en caso de que el color de dicho píxel se encuentre dentro del rango definido anteriormente. Luego, se obtiene su máscara complementaria (es decir, aquella que no es el fondo). Las coordenadas

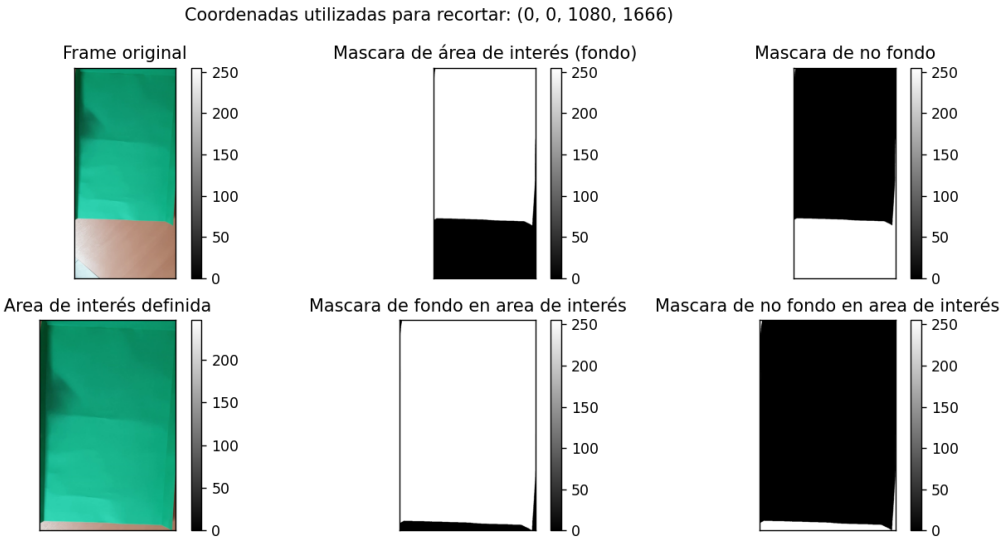

Figure 1.3: Detección área de interés

obtenidas en este primer frame serán las que se utilizarán a lo largo del análisis de todo el video para recortar el area de interés ya que el mismo no cambia, por lo que no es necesario recalcular las coordenadas del área de interés en cada frame sino que simplemente se recorta cada frame y sus máscaras por las coordenadas dadas en el primer frame.Esto también se realiza ya que de otra forma las áreas de interés cambiarían frame a frame. Dichas coordenadas son: (0, 0, 1080, 1666)

Este proceso se realiza con la siguiente función:

```python
def area_of_interest(frame:np.array, lower_limit: np.array, upper_limit: np.array, coords: tuple =
None)-> tuple[np.array, np.array, tuple]:
    frame_original_copy = frame.copy()
    frame_bg = cv2.cvtColor(frame_original_copy, cv2.COLOR_BGR2RGB)
    frame_hsv = cv2.cvtColor(frame_original_copy, cv2.COLOR_BGR2HSV)
    mask_bk = cv2.inRange(frame_hsv, lower_limit, upper_limit)

    mask_not_bk = cv2.bitwise_not(mask_bk)

    if coords:
        x, y, w, h = coords
    else:
        num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(mask_bk,
connectivity=8)
        areas = stats[:, cv2.CC_STAT_AREA]
        sorted_indices = np.argsort(areas)[::-1]
        sorted_stats = stats[sorted_indices]
        sorted_centroids = centroids[sorted_indices]
        sorted_stats = sorted_stats[1:]
        sorted_centroids = sorted_centroids[1:]
        fondo = sorted_indices[0]
        x, y, w, h = stats[fondo, cv2.CC_STAT_LEFT], stats[fondo, cv2.CC_STAT_TOP], stats[fondo,
cv2.CC_STAT_WIDTH], stats[fondo, cv2.CC_STAT_HEIGHT]

    area_interes = frame_bg[y:y+h, x:x+w]
    mask_not_bk_interes = mask_not_bk[y:y+h, x:x+w]
    mask_bk_interes = mask_bk[y:y+h, x:x+w]

    if coords:
        return area_interes, mask_not_bk_interes, mask_bk_interes,
    else:
        return area_interes, mask_not_bk_interes, mask_bk_interes, (x, y, w, h)
```

Figure 1.4: Función área de interés

1.3.2 Punto 2: Detectar frame donde los dados se dejan de mover.

Para esto, se recorren los frames del video y se realiza la diferencia frame a frame en escala de grises. Luego, si los píxeles que se movían eran menores a 0.0001 de los píxeles de dicho frame (en porcentaje, el 0.001), ese frame pasaba a ser "frame detenido". El frame donde se paran los dados es aquel frame que está detenido y sus 2 frames posteriores (como mínimo) también lo están. En el caso que detecta "frame detenido" se ve así:

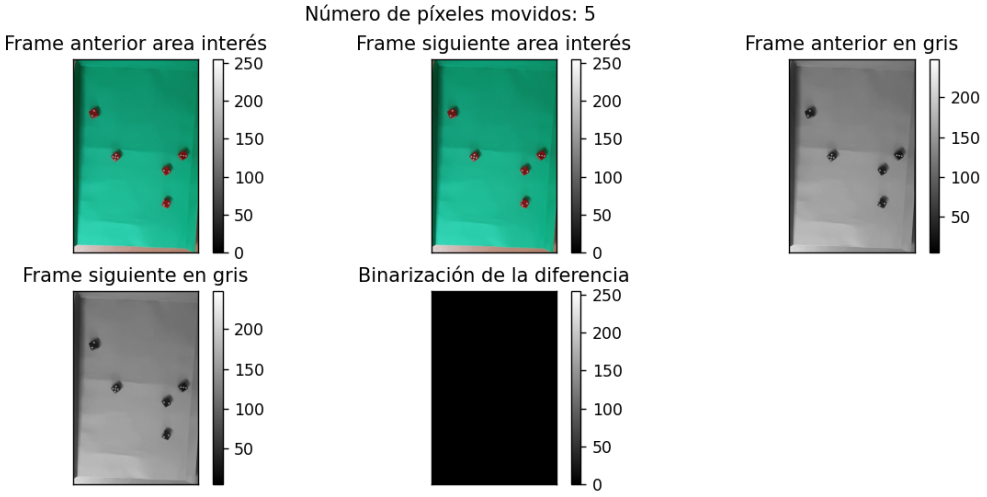

Figure 1.5: Frame detenido

En el caso que detecta frame en movimiento:

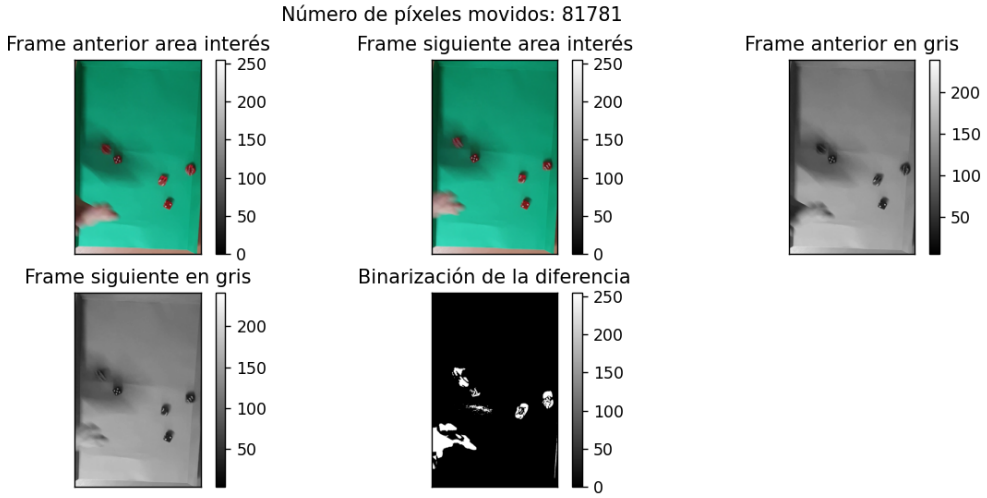

Figure 1.6: Frame movido

Una vez que encuentra un frame que cumpla con las condiciones dichas anteriormente para ser el frame donde se detienen los dados, se devuelve el número del frame.

Esto explicado en este punto es lo que hace la siguiente función:

```python
def find_frame_stop(video_path: str = "")-> int:
    # Abrir el video
    cap = cv2.VideoCapture(video_path)
    num_frame = 1
    frames_skip = 40
    movement = False
    consec_not_mov = 0
    frame_stop = 0
    ret, frame_ant = cap.read()

    while True:
        # Región de interés del primer frame
        frame_ant_interest, mask_bk_frame_ant, mask_not_bk_frame_ant = area_of_interest(frame_ant, low_li, up_li,
coords=coords_f)

        ret, frame_sig = cap.read()
        if not ret:
            break

        # Región de interés del siguiente frame
        frame_sig_interest, mask_bk_frame_sig, mask_not_bk_frame_sig = area_of_interest(frame_sig, low_li, up_li,
coords=coords_f)

        if num_frame > frames_skip:
            prop_mov = int(0.0001*frame_ant_interest.shape[0]*frame_ant_interest.shape[1])
            movement = stop(frame_ant_interest, frame_sig_interest, threshold=prop_mov)

            if movement:

                if consec_not_mov == 0:
                    frame_stop = num_frame
                consec_not_mov += 1
            else:
                consec_not_mov = 0


            # Si pasan 2 frames consecutivos sin movimiento, considerar que los dados se han detenido
            if consec_not_mov >= 2:

                break
        frame_ant = frame_sig

        num_frame += 1
```

Figure 1.7: Función stop.

1.3.3 Punto 3: Detectar los dados de todos los frames.

Para cada frame, se utilizan componentes conectadas a la máscara `not_bk` (es decir, lo que no es tablero) previamente recortada por las coordenadas del área de interés.

Se aplica una apertura con un kernel recto de tamaño $(21, 21)$ y dos iteraciones para separar dados que estén juntos en la máscara. Luego, se aplica un desenfoque gaussiano con un kernel de tamaño $(5, 5)$ y sigma igual a 0 para eliminar ruido (entendido como aquello que no es un dado).

Finalmente, se detectan las componentes conectadas y se filtran aquellas que no cumplan con las proporciones de un dado. Esto se realiza observando si el alto y ancho del *bounding box* de la componente son menores al 0.2 del alto y ancho correspondientes a la máscara, o si son menores al 0.05 del ancho de la máscara; si cumple con alguna de estas condiciones, no se considera un dado.

Se devuelve una lista donde cada elemento contiene las coordenadas de los dados (x, y, w, h).

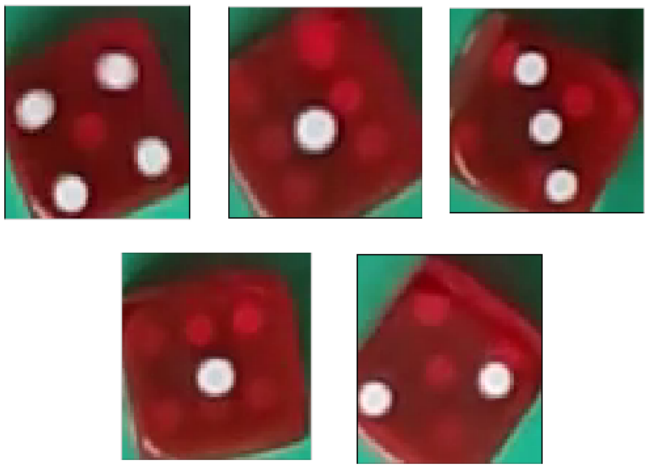

Figure 1.8: Dados detectados frame stop

Lo explicado anteriormente se hace con la función siguiente:

```python
def find_dados(frame_C,mask_not_bk: np.array)-> list:
    cuadrados = []
    k = cv2.getStructuringElement(cv2.MORPH_RECT, (21, 21))
    mask_open = cv2.morphologyEx(mask_not_bk, cv2.MORPH_OPEN, k, iterations= 2)
    mask_blur = cv2.GaussianBlur(mask_open, (5, 5), 0)
    _, mask_blur_binary = cv2.threshold(mask_blur, 170, 255, cv2.THRESH_BINARY)
    #imshow(mask_blur_binary)
    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(mask_blur_binary,
connectivity=4)
    for i in range(1, num_labels):
        x, y, w, h = stats[i, cv2.CC_STAT_LEFT], stats[i, cv2.CC_STAT_TOP], stats[i,
cv2.CC_STAT_WIDTH], stats[i, cv2.CC_STAT_HEIGHT]
        if h > (0.2* frame_C.shape[0]) or w > (0.2* frame_C.shape[1]) or w < (0.05 *
frame_C.shape[1]):
            continue
        cuadrado = frame_C[y:y+h, x:x+w]
        cuadrados.append((x,y,w,h))

    return cuadrados
```

Figure 1.9: Función para detectar dados

1.3.4 Punto 4: Detectar los puntos de los dados en el frame donde se detienen los dados.

Una vez ya obtenido el frame donde estan detenidos los dados y sus dados correspondientes, procedemos a contar los puntos por dados.

Para esto, hacemos uso de la lista de dados previa. Como se puede observar en la imagen de los dados del punto anterior, hay un factor indiscutible para detectar los puntos: el color blanco.

Así, se realiza una mascara que toma solo los valores blancos de una imagen (Con cierta flexibilidad). Definimos un rango de blanco en HSV desde [0,0,150] a [180,50,255] . Luego, con dicho rango sobre el recorte del dado y obtenemos una máscara que sólo contiene los valores blancos del dado. A la máscara le aplicamos un blur gaussiano para eliminar partes blancas que no son puntos del dado producto del rango de blanco permitido, y luego la binarizamos para terminar de eliminar dichas partes. Finalmente contamos las componentes conectadas de la máscara, obteniendo así los puntos de cada dado. El proceso se visualiza de la siguiente manera:

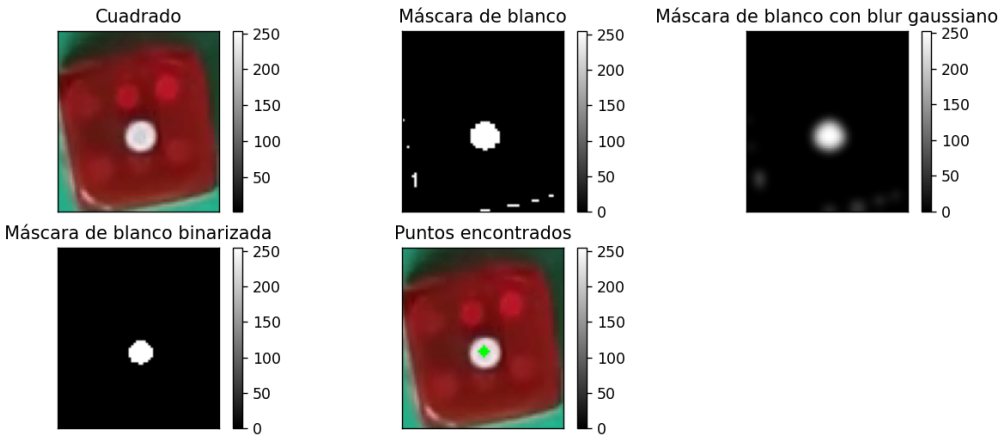

Figure 1.10: Puntos encontrados en el dado

Con estos puntos detectados para cada dado, ya obtenemos los valores de la jugada.
Todo lo explicado anteriormente, se realiza mediante esta función:

```python
def contar_puntos(frame, lista_dados: np.array)-> tuple[dict, int]:
    puntos_totales = 0
    dict_cuadrados_puntos = {}
    for i in range(len(lista_dados)):
        x,y,w,h = lista_dados[i]
        cuadrado = frame[y:y+h, x:x+w]
        cuadrado_hsv = cv2.cvtColor(cuadrado, cv2.COLOR_RGB2HSV)
        lower_white = np.array([0, 0, 150])
        upper_white = np.array([180, 50, 255])
        mask_white = cv2.inRange(cuadrado_hsv, lower_white, upper_white)
        mask_white_blur = cv2.GaussianBlur(mask_white, (11,11), 0)
        _, mask_white_blur_bin = cv2.threshold(mask_white_blur, 160, 255,
cv2.THRESH_BINARY)
        num_labels_c, labels_c, stats_c, centroids_c =
cv2.connectedComponentsWithStats(mask_white_blur_bin, connectivity=8)

        cuadrado_draw = cuadrado.copy()
        puntos = 0


        for k in range(1, num_labels_c):  #
            area_c = stats_c[k, cv2.CC_STAT_AREA]
            x_p, y_p, w_p, h_p = stats_c[k, cv2.CC_STAT_LEFT], stats_c[k,
cv2.CC_STAT_TOP], stats_c[k, cv2.CC_STAT_WIDTH], stats_c[k,
cv2.CC_STAT_HEIGHT]
            if h_p < (0.3 * cuadrado_draw.shape[0]) and w_p < (0.3 *
cuadrado_draw.shape[1]):
                cx, cy = int(centroids_c[k][0]), int(centroids_c[k][1])
                cv2.circle(cuadrado_draw, (cx, cy), 2, (0, 255, 0), -1)
                puntos += 1
            dict_cuadrados_puntos[(x,y,w,h)] = puntos

        puntos_totales += puntos

    return dict_cuadrados_puntos, puntos_totales
```

Figure 1.11: Función detectar puntos.

1.3.5 Punto 5: Detectar jugada en el frame donde se detienen los dados.

Para este punto, se utilizan los puntos por dados que se obtienen en el paso anterior. Se van viendo si las combinaciones de dados cumplen con algúna de las condiciones para hacer una jugada y luego se almacena la jugada obtenida con su correspondiente puntaje. Esto se realiza mediante la siguiente función:

```python
def geberala(combinaciones=list[int])-> tuple[int,str]:
    combinaciones.sort()  # Ordenar los valores para facilitar el análisis
    puntuacion_generala = 0
    text_combinacion = "Sin combinación"
    if len(combinaciones) == 5:
        # Generala
        if len(set(combinaciones)) == 1:
            puntuacion_generala = 50
            text_combinacion = "Generala: 50 puntos(5 dados iguales)"
        # Poker
        elif any(combinaciones.count(val) == 4 for val in combinaciones):
            puntuacion_generala = 40
            text_combinacion = "Poker: 40 puntos(4 dados iguales)"
        # Full
        elif any(combinaciones.count(val) == 3 for val in combinaciones):
            puntuacion_generala = 30
            text_combinacion = "Full: 30 puntos(3 dados iguales)"
        # Escalera
        elif combinaciones == [1, 2, 3, 4, 5]:
            puntuacion_generala = 20
            text_combinacion = "Escalera menor: 20 puntos"
        elif combinaciones == [2, 3, 4, 5, 6]:
            puntuacion_generala = 25
            text_combinacion = "Escalera mayor: 25 puntos"
        # Suma de números específicos
        else:
            puntuacion_generala = sum(combinaciones)  # Sumar los valores de
los dados
            text_combinacion = f"puntaje: {puntuacion_generala} puntos. No
forma juego"
    return puntuacion_generala, text_combinacion
```

Figure 1.12: Función generala.

1.3.6 Punto 6: Dibujar cada frame segun corresponda y mostrar el video.

Para este punto, necesitamos de tres funciones. La primera, dibuja los dados y también (si se lo indica) escribe los valores de los dados, la jugada obtenida y el puntaje. La segunda función une las áreas de interés modificadas con el frame original. Finalmente, la tercera función llama a la función anterior de forma distinta según el tipo de frame (stop, no stop, luego de jugada).

El video se procesa de la siguiente manera:

1. **Antes del "frame stop"**: En estos frames, solo se dibujan los rectángulos detectados en cada frame correspondiente.

2. **En el "frame stop"**: En este frame específico, se detectan los dados, la jugada y el puntaje. Estos valores se dibujan en pantalla.

3. **Si no se cambia la cantidad de dados = 5 en los frames siguientes**: Se siguen mostrando los valores obtenidos en el "frame stop" (dados, jugada y puntaje).

4. **Si se cambia la cantidad de dados = 5 en algún frame siguiente** : En el resto de los frames hasta el final del video solo se dibujan los rectángulos de los dados y se muestra la jugada con el puntaje detectados en el "frame stop".

Esto asegura que los valores importantes se muestren claramente en los momentos relevantes del video.

La primera función es la siguiente:

```python
def dibujar_dados(frame: np.array, lista_dados: list = None, dic_d: dict = None, label: bool = False,
puntaje_jugada: int = 0, texto: str = "Suma: 0 puntos")-> np.array:
    frame_draw = frame.copy()
    combinaciones = []
    puntuacion_generala = 0

    if dic_d is not None:
        i = 0
        for dado, punto in dic_d.items():
            x, y, w, h = dado
            combinaciones.append(punto)  # Agregar el valor del dado al análisis de combinaciones
            cv2.rectangle(frame_draw, (x, y), (x + w, y + h), (255, 0, 0), 2)
            if label:
                text_dado = f'Dado {i+1}'
                text_puntaje = f'Puntaje:{punto}'
                cv2.putText(frame_draw, text_dado, (x - 50, y - 65), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255),
thickness=2)
                cv2.putText(frame_draw, text_puntaje, (x - 50, y - 20), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255),
thickness=2)
                i += 1

        puntuacion_generala, text_combinacion = geberala(combinaciones)

        if label:
            text_puntaje = f'Puntaje total: {puntuacion_generala}'
            cv2.putText(frame_draw, text_combinacion, (20, 60), cv2.FONT_HERSHEY_DUPLEX, 1.8, (0, 255, 0),
thickness=2)

    else:
        for i in range(len(lista_dados)):
            x, y, w, h = lista_dados[i]
            cv2.rectangle(frame_draw, (x, y), (x + w, y + h), (255, 0, 0), 2)
        if puntaje_jugada != 0:
            text_puntaje = f'Puntaje total: {puntaje_jugada}'
            cv2.putText(frame_draw, texto, (20, 60), cv2.FONT_HERSHEY_DUPLEX, 1.8, (0, 255, 0), thickness=2)

    return frame_draw
```

Figure 1.13: Función dibujar dados.

En cuanto a la segunda función, es esta:

```python
def unir_frames(frame_original:np.array, area_dibujada:np.array, coords: tuple)-> np.array:
    frame_dibujado = frame_original.copy()
    x, y, w, h = coords
    frame_dibujado[y:y+h, x:x+w] = area_dibujada
    return frame_dibujado
```

Figure 1.14: Función unir frames.

La tercera función mencionada es la siguiente y hace uso de la segunda función para hacer los frames modificados:

```python
def modificar_frames(video_path, frame_stop, low_limit, up_limit, coords, new_w, new_h) -> list:
    cap = cv2.VideoCapture(video_path)
    ret, frame_ant = cap.read()
    if not ret:
        print("No se pudo leer el video.")
        return []

    # Inicialización de propiedades del video
    num_frame = 0
    frames_mod = []
    total_puntos = 0
    move = False

    while True:
        ret, frame_sig = cap.read()
        if not ret:
            break

        # Procesamiento según la lógica del frame actual
        if num_frame < frame_stop:
            area_interes_f_n, mask_not_bk_f_n, mask_bk_f_n = area_of_interest(
                frame_sig, lower_limit=low_limit, upper_limit=up_limit, coords=coords
            )
            area_interes_f_n = cv2.cvtColor(area_interes_f_n, cv2.COLOR_BGR2RGB)
            lista_dados = find_dados(area_interes_f_n, mask_not_bk_f_n)
            area_dib = dibujar_dados(area_interes_f_n, lista_dados=lista_dados)
            frame_dib = unir_frames(frame_sig, area_dib, coords=coords)

        elif num_frame == frame_stop:
            area_interest_frame_stop, mask_not_bk_frame_stop, mask_bk_frame_stop = area_of_interest(
                frame_sig, lower_limit=low_limit, upper_limit=up_limit, coords=coords
            )
            lista_dados = find_dados(area_interest_frame_stop, mask_not_bk_frame_stop)
            dic_dados, puntaje = contar_puntos(area_interest_frame_stop, lista_dados)
            combinaciones = [punto for _, punto in dic_dados.items()]
            puntaje_generala, texto_ = generala(combinaciones)
            texto = texto_
            total_puntos += puntaje

            area_interest_frame_stop = cv2.cvtColor(area_interest_frame_stop, cv2.COLOR_BGR2RGB)
            area_dib = dibujar_dados(area_interest_frame_stop, dic_d=dic_dados, label=True, puntaje_jugada=puntaje)
            frame_dib = unir_frames(frame_sig, area_dib, coords=coords)

        elif frame_stop < num_frame and not move:
            area_interest_frame_stop, mask_not_bk_frame_stop, mask_bk_frame_stop = area_of_interest(
                frame_sig, lower_limit=low_limit, upper_limit=up_limit, coords=coords
            )
            lista_dados = find_dados(area_interest_frame_stop, mask_not_bk_frame_stop)
            if len(lista_dados) < 5:
                move = True
                continue
            area_interest_frame_stop = cv2.cvtColor(area_interest_frame_stop, cv2.COLOR_BGR2RGB)
            area_dib = dibujar_dados(area_interest_frame_stop, dic_d=dic_dados, label=True, puntaje_jugada=puntaje)
            frame_dib = unir_frames(frame_sig, area_dib, coords=coords)

        if move:
            area_interes_f_n, mask_not_bk_f_n, mask_bk_f_n = area_of_interest(
                frame_sig, lower_limit=low_limit, upper_limit=up_limit, coords=coords
            )
            lista_dados = find_dados(area_interes_f_n, mask_not_bk_f_n)
            area_interes_f_n = cv2.cvtColor(area_interes_f_n, cv2.COLOR_BGR2RGB)
            area_dib = dibujar_dados(area_interes_f_n, lista_dados=lista_dados, puntaje_jugada=puntaje_generala, texto=texto)
            frame_dib = unir_frames(frame_sig, area_dib, coords=coords)

        num_frame += 1
        frame_dib_resize = cv2.resize(frame_dib, (new_w, new_h))
        cv2.imshow(winname='', mat=frame_dib_resize)

        # Salir si el usuario presiona 'q'
        if cv2.waitKey(30) & 0xFF == ord('q'):
            print("Detenido por el usuario.")
            break

        frames_mod.append(frame_dib)

    # Liberar recursos
    cap.release()
    cv2.destroyAllWindows()

    print("Procesamiento completado.")
    return frames_mod, dic_dados, puntaje_generala, total_puntos
```

Figure 1.15: Función modificar frames.

1.3.7 Punto 7: Guardar el video.

Para guardar el video, se utiliza la lista de frames modificados que se obtiene del paso anterior y un path output.

```python
def guardar_video(lista_frames, output_path, fps=30, codec="mp4v"):
    if not lista_frames:
        raise ValueError("La lista de frames está vacía. No se puede crear un video.")

    # Crear directorio si no existe
    os.makedirs(os.path.dirname(output_path), exist_ok=True)


    height, width, _ = lista_frames[0].shape


    fourcc = cv2.VideoWriter_fourcc(*codec)
    out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

    # Escribir cada frame en el archivo de video
    for frame in lista_frames:
        out.write(frame)


    out.release()

    print(f"Video guardado correctamente en: {output_path}")
```

Figure 1.16: Función guardar video.

1.4 Resultados obtenidos.

Video 1:

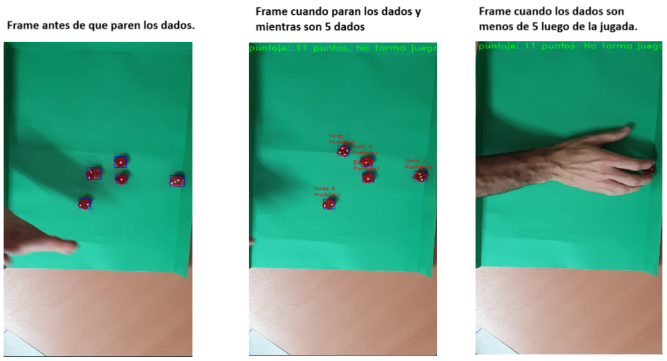

Figure 1.17: Resultado video 1

Video 2:

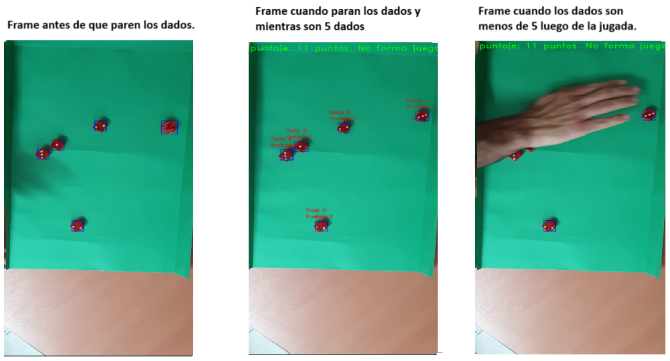

Figure 1.18: Resultado video 2

Video 3:

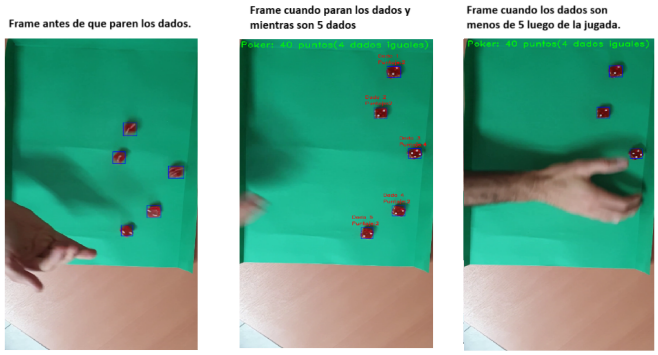

Figure 1.19: Resultado video 3

Video 4:

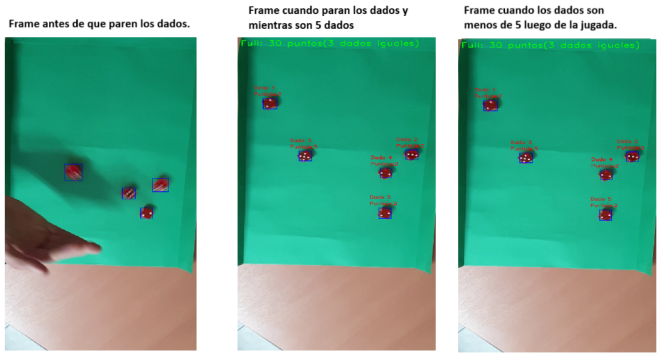

Figure 1.20: Resultado Video 4

1.5 Requerimientos

Además de python 3.11 o superior:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact, IntSlider
from PIL import Image
import os
```

Figure 1.21: Requerimientos

1.6 Modo de uso.

Ejecute el programa con python Le solicitará la ubicación del archivo y la ubicación donde desea (si

```
py .\generala.py
```

Figure 1.22: Ejecutar programa.

es que lo quiere) guardar el video modificado: El programa le mostrará el video y una salida como

```
Ingrese la ubicación del archivo: data\tirada_3.mp4
Ingrese la ubicación donde desea guardar el video. Si no
desea guardarlo, presione enter: output\tirada_2_mod.mp4
```

Figure 1.23: Solicitud programa.

la siguiente por consola:

```
Procesando video data\tirada_3.mp4 ...
Procesamiento completado.
Video guardado correctamente en: output\tirada_3_mod.mp4
Esta tirada tuvo los siguientes puntos:
El dado 1: 2
El dado 2: 2
El dado 3: 4
El dado 4: 2
El dado 5: 2
El total de puntos es: 12
El puntaje total fue: 40
```

Figure 1.24: Salida programa

1.7 Guía funciones.

Se muestran las funciones utilizadas con una breve explicación:

FUNCIONES UTILIZADAS.		
FUNCIÓN.	EXPLICACIÓN.	PARÁMETROS
green_sample(frame)	Toma un frame y realiza una muesta de color verde a partir de la cual devuelve límites inferiores y superiores de rango de dicho color que sirve para hacer máscaras.	- frame: frame del cual se obtiene la muestra.
area_of_interest(frame, lower_limit, upper_limit, coords)	Toma un frame y obtiene una máscara de color verde y su complementaria dentro de coordenadas de interés (si las recibe) a partir de un límite inferior y superior dados.	- frame: frame del cual se obtiene la muestra. - upper_limmit: límite superior del color verde. - lower_limit: límite inferior del color verde. - coords: coordenadas de interés.
stop(frame_ant, frame_sig, threshold)	Devuelve el True o False según si no se sobrepasa de un threshold de movimiento de un frame a otro.	- frame_ant: frame_anterior. - frame_sig: frame siguiente. - threshold: umbral de movimiento.
find_frame_stop(video_path)	Esta función toma un video y devuelve el número de frame donde se detiene el movimiento dentro de un área de interés predefinida.	- video_path: ubicación del video.
find_dados(frame_C,mask_not_bk)	Obtiene los rectángulos de las componentes conectadas presentes en la máscara no fondo de un frame dado y devuelve aquellos que por su forma considera que son dados.	- frame_c: frame en RGB. - mask_not_bk: máscara que no es fondo del frame.
contar_puntos(frame, lista_dados)	Cuenta los puntos de cada dado dentro de un frame.	- frame: frame en RGB. - lista_dados: lista de dados en un frame.

Figure 1.25: Guía funciones parte 1

FUNCIONES UTILIZADAS.		
FUNCIÓN.	EXPLICACIÓN.	PARÁMETROS
generala(combinaciones)	Detecta la jugada obtenida (o no) de una tirada mediante el análisis de los puntos de los dados en la misma.	- combinaciones: lista de enteros que representan los valores de los dados.
dibujar_dados(frame, lista_dados, dic_d, label, puntaje_jugada, texto)	Dibuja los dados de acuerdo a si se pasa una lista_dados o un dic_d.	- frame: frame en RGB. - lista_dados: lista de dados. - dic_d: diccionario dado-valor. - label: indica si se quiere o no dibujar los labels en caso de que se pase un dic_d. - puntaje_jugada: puntaje de la jugada si es que se pasa. - texto: texto con el puntaje de la jugada.
unir_frames(frame_original, area_dibujada, coords)	Une el frame original con el frame dibujado.	- frame_original: frame RGB sobre el cual se quiere unir el área dibujada. - area_dibujada: área dibujada. - coords: coordenadas para unir sobre el frame original el área dibujada.
modificar_frames(video_path, frame_stop, low_limit, up_limit, coords, new_w, new_h)	Modifica los frames según el momento de la jugada en que se encuentren. Muestra el video modificado.	- video_path: path del video que contiene la jugada. - frame_stop: número de frame donde se detienen los dados. - up_limit: límite superior de verde. - coords: coordenadas del área de interés del video. - new_h: nueva altura del video (sólo para fines de visualización). - new_w: nuevo ancho del video (sólo para fines de visualización).

Figure 1.26: Guía funciones parte 2

FUNCIONES UTILIZADAS.		
FUNCIÓN.	EXPLICACIÓN.	PARÁMETROS
guardar_video(lista_frames, output_path, fps=30, codec="mp4v")	Guarda un video en 30 fps y formato mp4 a partir de una lista de frames.	- lista_frames: lista de frames a partir de la cual se arma el video. - output_path: ubicación donde guardar el video. - fps: fps del video, seteado a 30. - codec: codificación del video, seteada a "mp4v" (.mp4).
exec(path, output_video_path)	Ejecuta las funciones necesarias para detectar la jugada y dibujar el video.	- path: path del video. - output_video_path: ubicación de salida del video.

Figure 1.27: Guía funciones parte 3